E3SWE: Webservice med docker-compose

Janus Bo Andersen
JA67494
Ingeniørhøjskolen i Aarhus, Campus Herning

12. december 2019

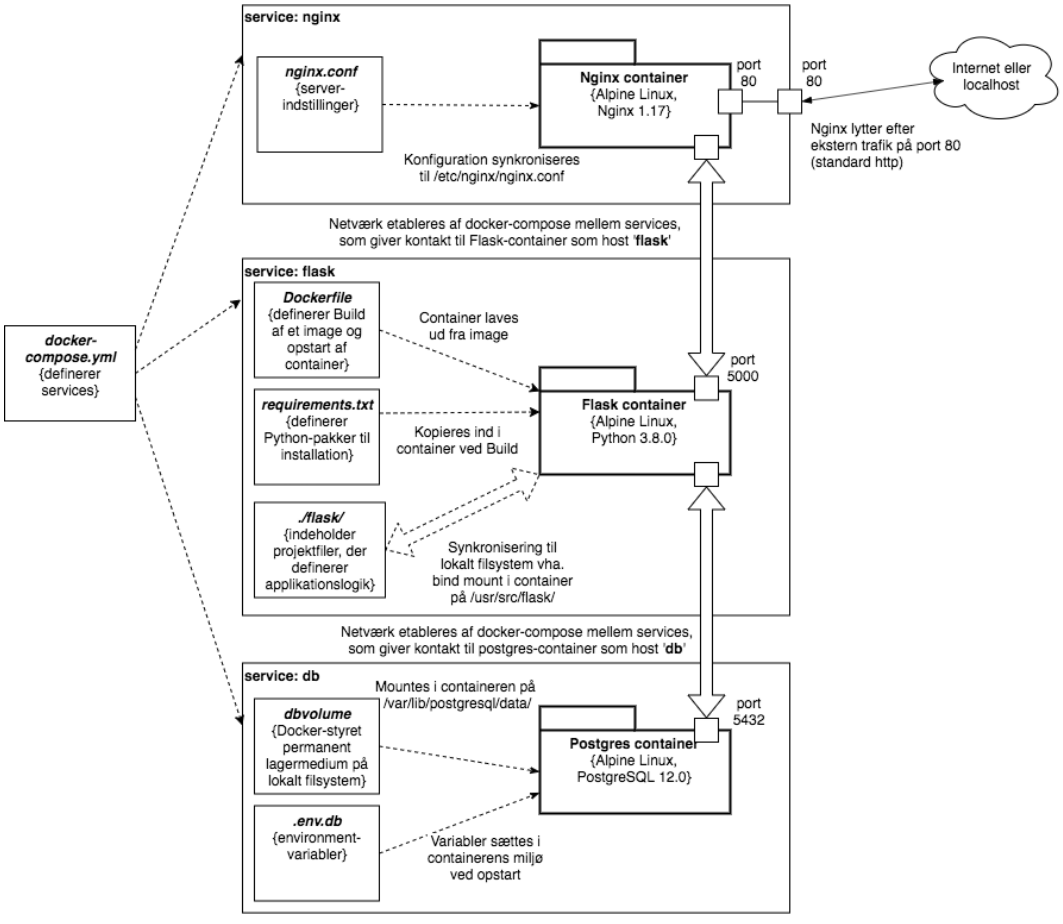
1 Indledning

Denne journal beskriver, hvordan en basal webservice sættes op vha. docker-compose. Webservicen skal have en frontend/proxy-server (Nginx), applikationslogik og applikationsserver (Flask) og en database-server (PostgreSQL). Den skal kunne tilgås på port 80 (standard Http). Der skal tilknyttes et permanent lagermedium (Docker volume) til databasen.

En tilsvarende webservice er sat op af forfatteren m.fl. ifm. E3PRO3, dog med Django i stedet for Flask [1]. Al kode samt dette dokument kan ses på forfatterens Github [2].

2 Design

Følgende figur illustrerer design af systemet. Images er udvalgt fra officielle images på Docker Hub [3].



Figur 1: Design og implementering (egen tilvirkning)

3 Implementering

Det er værd at bemærke princippet, at *specifikke* versioner af images er valgt og fastlåst i implementering. Implementerede versioner af images ses i figur 1. I `docker-compose.yml`, er oprettelse af netværk (bridges) sat til at foretages automatisk af docker-compose, der er derfor ikke navngivne netværksbroer. Alle containers kan forbindes via deres service/host-navne. Endelig `docker-compose.yml`:

```yaml
version: '3.7'

services:
  flask:
    build: ./flask
    volumes:
      - ./flask/:/usr/src/flask/
    depends_on:
      - db
    command: >
      env FLASK_APP=app.py flask run --host=0.0.0.0 --port=5000

  db:
    image: postgres:12.0-alpine
    restart: always
    volumes:
      - dbvolume:/var/lib/postgresql/data/
    env_file:
      - ./.env.db

  nginx:
    image: nginx:1.17-alpine
    depends_on:
      - db
    restart: always
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
    ports:
      - "80:80"

volumes:
  dbvolume:
```

Figur 2: Dockerfile

`Dockerfile` til Flask-app'en er også gengivet her, fordi den er "custom".

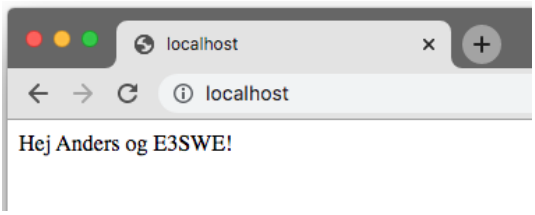
```
#Pull image for Linux Alpine with Python 3.8
FROM python:3.8.0-alpine
MAINTAINER Janus Bo Andersen

#Set the working directory inside the Alpine Container
WORKDIR /usr/src/flask

#Set environment variables
#don't write .pyc files to disk
ENV PYTHONDONTWRITEBYTECODE 1
#don't buffer stdout, as that causes errors
ENV PYTHONUNBUFFERED 1

#Install dependencies for PostgreSQL (Psycopg2)
RUN apk update && apk add postgresql-dev gcc musl-dev shadow curl

#Install dependencies inside container
RUN pip install --upgrade pip
COPY ./requirements.txt /usr/src/flask/requirements.txt
RUN pip install -r requirements.txt

#Create a user to avoid running as root.
RUN adduser -D nonrootuser
USER nonrootuser
```

Figur 3: Dockerfile

Systemet startes med `docker-compose up`. Switch `-d` bruges for at køre services i baggrunden, og `--build` for at tvinge et rebuild af alle containers.

4 Test

Koden er testet i Chrome på Mac med Docker Desktop ver. 2.1.0.4, Docker Engine ver. 19.03.4 og docker-compose 1.24.1. Følgende figur viser viser OK testresultat for request på localhost:80.



Figur 4: Testresultat: Browser

Næste figur viser, at Nginx responderer på en Http-request, og proxyer til Flask.

```
Starting system_db_1 ... done
Starting system_nginx_1   ... done
Recreating system_flask_1 ... done
Attaching to system_db_1, system_nginx_1, system_flask_1
db_1     | 2019-12-11 20:06:10.752 UTC [1] LOG:  starting PostgreSQL 12.0 on x86_64-pc-linux-musl, compiled by gcc (Alpine 8.3.0) 8.3.0, 64-bit
db_1     | 2019-12-11 20:06:10.752 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
db_1     | 2019-12-11 20:06:10.752 UTC [1] LOG:  listening on IPv6 address "::", port 5432
db_1     | 2019-12-11 20:06:10.761 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
db_1     | 2019-12-11 20:06:10.784 UTC [19] LOG:  database system was shut down at 2019-12-11 20:04:54 UTC
db_1     | 2019-12-11 20:06:10.790 UTC [1] LOG:  database system is ready to accept connections
flask_1  |  * Serving Flask app "app.py"
flask_1  |  * Environment: production
flask_1  |    WARNING: This is a development server. Do not use it in a production deployment.
flask_1  |    Use a production WSGI server instead.
flask_1  |  * Debug mode: off
flask_1  |  * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
nginx_1  | 172.23.0.1 - - [11/Dec/2019:20:06:48 +0000] "GET / HTTP/1.1" 200 20 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6) AppleWebKit/5
flask_1  | 172.23.0.3 - - [11/Dec/2019 20:06:48] "GET / HTTP/1.0" 200 -
```

Figur 5: Testresultat: Services

Systemet fungerer som ønsket.

5 Forbedringsforslag

- Flask's applikationsserver er ikke til produktion, og bør udskiftes med en WSGI-server med produktionskapacitet/sikkerhed, fx gunicorn.
- Databasen bliver ikke pt. benyttet af Flask. For at det kan virke godt, skal requirements.txt indeholde:
  - psycopg2 (driver/adapter til PostgreSQL), og så bør postgresql-dev installeres via systemets package manager (APK).
  - flask_sqlalchemy (el.lign. toolkit/ORM til OOP).
  - flask_migrate (til at implementere schema-ændringer i databasen).
- Opsætning så der kan serves staticfiles.
- Opsætning af SSL/TLS-krypteret trafik (Https på port 443).
- Klargøring til deployment, fx til Heroku, AWS eller lignende.
- Noget *langt mere* interessant indhold.

6 Konklusion

Denne journal har kort dokumenteret, hvordan en basal webservice sættes op. Der er desuden givet et par forbedringsforslag til webservicen. Al kode er tilgængelig på Github.

Litteratur

[1] E3PRO3 Team 2, 2019. *E3PRO3 Server setup*, url: https://github.com/AUTeam2/server-setup.

[2] Janus Bo Andersen. *Github repo e3swe_exam*, url: https://github.com/janusboandersen/e3swe_exam.

[3] Docker Hub. *Official Docker Images*, url: https://hub.docker.com/search?type=image&image_filter=official.